Brief Contributions

Pfair Scheduling of Generalized Pinwheel Task Systems

Sanjoy K. Baruah, *Member*, *IEEE*, and Shun-Shii Lin

Abstract—The scheduling of generalized pinwheel task systems is considered. It is shown that pinwheel scheduling is closely related to the fair scheduling of periodic task systems. This relationship is exploited to obtain new scheduling algorithms for generalized pinwheel task systems. When compared to traditional pinwheel scheduling algorithms, these new algorithms are both more efficient from a run-time complexity point of view, and have a higher *density threshold,* on a very large subclass of generalized pinwheel task systems.

Index Terms—Generalized pinwheels, fairness, real-time scheduling, density threshold.

1 Introduction

THE pinwheel scheduling problem was introduced by Holte et al. [15] in the context of offline scheduling for satellite-based communication. The problem is defined as follows: *Given a multiset* $\{b_1, b_2, \ldots, b_n\}$ *of positive integers, determine an infinite sequence* (a schedule) *over the symbols* $\{1, 2, 3, \ldots, n\}$ *such that, for each* i, $1 \le i \le n$, *any subsequence of* b_i *consecutive symbols contains at least one* i. Since its introduction, this task model has been used to model the requirements of a wide variety of real-time systems. For example, Han and Lin [12] used pinwheel techniques to model distance-constrained tasks; Hsueh et al. [17] extended this research to distributed systems. Baruah et al. [6] used pinwheel scheduling to construct static schedules for sporadic task systems. Han and Shin [13], [14] applied pinwheel techniques to real-time network scheduling. Recently, Baruah and Bestavros [4] modeled fault-tolerance and real-time requirements of broadcast disks by generalizing the pinwheel model.

For all of these applications, the original model of Holte et al. [15] has been generalized as follows: *Given a multiset* $\{(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)\}$ *of ordered pairs of positive integers, determine an infinite sequence over the symbols* $\{1, 2, 3, \ldots, n\}$ *such that, for each* i, $1 \le i \le n$, *any subsequence of* b_i *consecutive symbols contains at least* a_i *is*. The a_is typically represent computation requirements of tasks, or the amount of data to be transferred, and are, hence, often significantly greater than one. Of course, any generalized pinwheel task system $\{(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)\}$ can be transformed to the original model in a straightforward manner by simply representing each ordered pair (a_i, b_i) as a sequence of a_i distinct b_is. However, such a transformation will result in a multiset of size $a_1 + a_2 + \ldots + a_n$ integers; in general, this represents an exponential increase in the size of the representation. As a consequence, the algorithms used for scheduling pinwheel task systems represented in the original model are not necessarily the most efficient ones for scheduling generalized pinwheel task systems.

- *S.K. Baruah is with the Department of Computer Science, University of Vermont, Burlington, VT 05405. E-mail: sanjoy@cs.uvm.edu.*
- *S.-S. Lin is with the Department of Information and Computer Education, National Taiwan Normal University, 162 Hoping East Road, Sec. 1, Taipei, Taiwan, 10610, R.O.C. E-mail: linss@ice.ntnu.edu.tw.*

Manuscript received 6 June 1997.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 102118.

In this research, we address the issue of scheduling generalized pinwheel task systems. We establish a close link between pinwheel scheduling and recent research on the *fair* scheduling of periodic tasks [3], [5], [7] and exploit this relationship to determine a new scheduling strategy for generalized pinwheel task systems. Apart from proving more efficient from a run-time complexity point of view, our approach proves superior to algorithms currently used for scheduling pinwheel task systems in terms of *schedulability*—the ability to schedule a large class of pinwheel task systems—for generalized pinwheel task systems in which all the a_i parameters are relatively large.

The rest of this paper is organized as follows. In Section 2, we formally define generalized pinwheel task systems and briefly review some of the previous results in pinwheel scheduling theory. In Section 3, we review the recent research in fair scheduling of periodic task systems, particularly the notions of *proportionate progress* and *pfairness*. In Section 4, we present and prove correct Algorithm Pinfair, an algorithm for scheduling generalized pinwheel task systems that is directly based upon a fair scheduling algorithm. In Section 5, we evaluate the behavior of Algorithm Pinfair by comparing it to other approaches to pinwheel scheduling.

2 Pinwheel Task Systems

Consider a shared resource that is to be scheduled in accordance with the *integral boundary constraint*: For each integer $t \ge 0$, the resource must be allocated to exactly one task (or remain unallocated) over the entire time interval $[t, t + 1)$. (We refer to this time interval as *time slot* t.) For our purposes, a **pinwheel task** x is characterized by a *computation requirement* $x.a \in \mathbf{N}$ and a *window size* $x.b \in \mathbf{N}$, with the interpretation that the task x expects to be allocated the shared resource for at least $x.a$ out of every $x.b$ consecutive time slots. A *pinwheel task system* is a set of pinwheel tasks that share a single resource.

The ratio of the computation requirement of a task to its window size is referred to as the *density* of the task. The density of a system of tasks is simply the sum of the densities of all the tasks in the system. Observe that, for a task system to be schedulable, it is necessary (although *not* sufficient) that the density of the system not exceed one.

The issue of designing efficient scheduling algorithms for pinwheel task systems has been the subject of much research. Holte et al. [16] presented an algorithm which schedules any pinwheel task system of two tasks with density at most one. Lin and Lin [18] have designed an algorithm which schedules any pinwheel task system of three tasks with a density at most five-sixths (this algorithm is optimal in the sense that there are three-task systems with density $5/6 + \epsilon$ that are infeasible, for ϵ arbitrarily small). When the number of tasks is not restricted, Holte et al. [15] have a simple and elegant algorithm for scheduling any pinwheel task system with density at most one-half. Chan and Chin [8] have improved this result and designed an algorithm for efficiently scheduling any pinwheel system with a density at most 0.7.

3 Proportionate Progress

A **periodic task** x is characterized by a *period* $x.p \in \mathbf{N}$ and a *computation requirement* $x.e \in \mathbf{N}$, with the interpretation that the task x expects to be allocated the processor for $x.e$ units of time in every interval $\{t \mid i \cdot x.p \le t < (i + 1) \cdot x.p\}$, for each $i \in \mathbf{N}$. Given an instance Φ of n such periodic tasks, the (uniprocessor) **periodic scheduling problem** [20] is concerned with attempting to schedule these n tasks on a single processor so as to satisfy the constraints of each task. Task preemption is permitted, but only at integral boundaries as dictated by the integral boundary constraint.

Liu and Layland have shown [20] that $\sum_{x \in \Phi}(x.e/x.p) \le 1$ is a necessary and sufficient condition for a system Φ of periodic tasks to have a periodic schedule; furthermore, the *earliest deadline first* scheduling algorithm (EDF) [10] has been proven an optimal scheduling algorithm.

3.1 Temporal Fairness

The issue of *fairness* in resource-allocation and scheduling has recently been attracting considerable attention [1], [2], [5]. Motivated no doubt in part by applications, such as multimedia, which are characterized by fairly "regular" resource requirements over extended intervals, attempts have been made to formalize and characterize notions of temporal fairness. The concepts of *proportionate progress* and *pfairness* were introduced in [5] (see also [7]) to quantitatively measure the fairness of a schedule. We briefly review these ideas below:

We start with some conventions:

- We adopt the standard notation of having $[a, b)$ denote the contiguous natural numbers $a, a + 1, \ldots, b - 1$.
- The real interval between time t and time $t + 1$ (including t, excluding t+1) will be referred to as slot t, $t \in \mathbf{N}$.
- We will consider an instance that involves one processor and a set Φ of n tasks.
- Each task x has two integer attributes—a *period* $x.p$ and an *execution requirement* $x.e$. We define the *weight* $x.w$ of task x to be the ratio $x.e/x.p$. Furthermore, we assume $0 < x.w \le 1$.

A *schedule* S for instance Φ is a function from the natural numbers $\{0, 1, 2, \ldots\}$ to $\Phi \cup \{\perp\}$, with the interpretation that $S(t) = x$, $x \in \Phi$, if the processor is allocated to task x for slot t, and $S(t) = \perp$ if the processor is unallocated during time-slot t. With respect to a particular schedule S (which should be clear from context), let $\text{allocated}_x(t)$ be defined as follows:

$$\text{allocated}_x(t) \stackrel{\text{def}}{=} \left|\left\{t' \,\middle|\, t' \in [0, t) \text{ and } S(t') = x\right\}\right|.$$

Schedule S is a *periodic schedule* if and only if

$$\forall i, x : i \in \mathbf{N}, x \in \Phi : \text{allocated}_x(i \cdot x.p) = x.e \cdot i.$$

That is, each task x is allocated exactly $i \cdot x.e$ slots during its first i periods, for all i.

Let us define the *lag* of a task x at time t with respect to schedule S, denoted $\text{lag}(S, x, t)$, as follows:

$$\text{lag}(S, x, t) \stackrel{\text{def}}{=} x.w \cdot t - \text{allocated}_x(t).$$

The quantity $x.w \cdot t$ represents the amount of time for which task x *should* have been allocated the processor over $[0, t)$.

Informally, a schedule displays proportionate progress (equivalently, satisfies *pfairness*, or is *pfair*) if at all integer time instants t and for all tasks x, the lag of task x at time t—the difference between the amount of time for which x should have been allocated a processor and the amount of time for which it was allocated a processor—is strictly less than 1 in absolute value. More formally, schedule S is *pfair* if and only if

$$\forall x, t : x \in \Phi, t \in \mathbf{N} - 1 < \text{lag}(S, x, t) < 1.$$

That is, a schedule is pfair if and only if it is never the case that any task x is overallocated or underallocated by an entire slot.

Pfairness is an extremely stringent form of fairness—indeed, it has been shown [5] that no stronger fairness can be guaranteed achievable for periodic task systems in general. (Consider a system of n identical tasks, each with weight $1/n$. The task that is scheduled at slot 0 has a lag $(-1 + 1/n)$ at time 1, and the one scheduled at slot $n - 1$ has a lag $(1 - 1/n)$ at time $(n - 1)$. By making n large, these lags can be made arbitrarily close to –1 and +1, respectively.) It was proven in [5] that pfair scheduling is a stronger requirement than periodic scheduling, in that any pfair schedule is periodic. The converse, however, is not generally true.

Equations (1) and (2) (below) further characterize the structure of pfair schedules. With respect to a given task x, let $\text{earliest}(x, j)$ (resp., $\text{latest}(x, j)$) denote the earliest (resp., latest) slot during which x may be scheduled for the jth time, $j \in \mathbf{N}$, in any pfair schedule. We can easily derive closed-form expressions for $\text{earliest}(x, j)$ and $\text{latest}(x, j)$:

$$\text{earliest}(x, j) = \min t : t \in \mathbf{N} : x.w \cdot (t + 1) - (j + 1) > -1 = \left\lfloor \frac{j}{x.w} \right\rfloor. \quad (1)$$

Similarly,

$$\text{latest}(x, j) = \max t : t \in \mathbf{N} : x.w \cdot t - j < 1 = \left\lceil \frac{j + 1}{x.w} \right\rceil - 1. \quad (2)$$

A slot t such that $t = \text{earliest}(x, j)$ for some j is called an **activation-slot** for task x.

With respect to a particular schedule S,

- Task x is **contending** at time t if it may receive the processor without becoming overallocated; i.e., if
$$\text{allocated}_x(t) = k \wedge \text{earliest}(x, k) \le t.$$
- For contending tasks, a **pseudodeadline** is defined: The pseudodeadline of task x is the time by which x must be allocated the processor if it is to not violate its lag constraints. More formally, for contending task x at time t, the **pseudodeadline**
$$x.d \stackrel{\text{def}}{=} \text{latest}(x, \text{allocated}_x(t)).$$

3.2 Algorithm PF

The concept of pfairness was introduced in [5], in the context of constructing periodic schedules for a system of periodic tasks on several identical processors—the *multiprocessor periodic scheduling problem* [19]. The following theorem was proven there, by means of some fairly involved network-flow constructions, and by using the Integer Flow Theorem [11]:

THEOREM 1. *A system of periodic tasks can be scheduled in a pfair manner on m processors provided the weights of all the tasks sum to at most m.*

As a special case, we obtain the following corollary with respect to *uni*processor systems:

COROLLARY 1.1. *Every uniprocessor system of periodic tasks* Φ *for which* $\left(\sum_{x \in \Phi} x.w \le 1\right)$ *holds has a pfair schedule.*

In addition, an on-line scheduling algorithm—Algorithm PF—was presented and proven correct. This algorithm has a nontrivial priority scheme that requires $O\left(\sum_{\text{all } x} \lceil \log(x.p + 1) \rceil\right)$ time to determine the m highest-priority tasks in the worst case. However, for the uniprocessor case, Algorithm PF reduces to simply allocating the processor at each time slot to the highest-priority contending task according to the following pseudo-deadline based priority rule:

contending task x has priority over contending task y iff $x.d \le y.d$*—ties broken arbitrarily.*

Again with respect to the uniprocessor case, Algorithm PF can be implemented using the heap-of-heaps data structure [21] in $O(\log n)$ time per time slot, where n is the number of tasks.

EXAMPLE 1. Consider a system of two periodic tasks $x_1.e = 6$, $x_1.p = 10$ and $x_2.e = 3$, $x_2.p = 9$, which are to be scheduled on a single processor. The initial portion of the schedule generated for this task system by Algorithm PF is given below (an initial trace of the computations performed by Algorithm PF

TABLE 1
PARTIAL TRACE OF THE COMPUTATIONS PERFORMED BY ALGORITHM PF ON THE EXAMPLE TASK SYSTEM OF EXAMPLE 1

Slot	0	1	2	3	4	5	…
task	1	2	1	1	2	1	…
$t \cdot x_1.w$	0	0.6	1.2	1.8	2.4	3	…
$k_1 = \text{allocated}_{x_1}(t)$	0	1	1	2	3	3	…
lag(S, x_1, t)	0	–0.4	0.2	–0.2	–0.6	0	…
earliest(x_1, k_1)	0	1	1	3	5	5	…
$x_1.d$ = latest(x_1, k_1)	1	3	3	4	6	6	…
$t \cdot x_2.w$	0	1/3	2/3	1	4/3	5/3	…
$k_2 = \text{allocated}_{x_2}(t)$	0	0	1	1	1	2	…
lag(S, x_2, t)	0	1/3	–1/3	0	1/3	–1/3	…
earliest(x_2, k_2)	0	0	3	3	3	6	…
$x_2.d$ = latest(x_2, k_2)	2	2	5	5	5	8	…

on this task system is given in Table 1—the reader wishing a deeper understanding of Algorithm PF is encouraged to work through this trace):

slot:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
task:	1	2	1	1	2	1	1	2	1	2	1	1	2	1	$\perp$
slot:	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
task:	1	2	1	1	2	1	1	2	1	2	1	1	2	1	$\perp$

The reader may verify that this schedule has indeed been generated according to Algorithm PF. (For example, at time $t = 10$, $\text{allocated}_{x_2}(10) = 4$, while $10 \times x_2.w = 3.\overline{3}$; hence,

$$\text{lag}(S, x_2, 10) = -0.\overline{6}.$$

Since $earliest(x_2, 4) = \lfloor 4/x_2.w \rfloor = 12$, x_2 is not contending at this time. However, $\text{allocated}_{x_1}(10) = 6$, while

$$\text{earliest}(x_1, 6) = \lfloor 6/x_1.w \rfloor = 10;$$

hence, x_1 is contending at $t = 10$, and

$$x_1.d = \text{latest}(x_1, 6) = \lceil 7/x_1.w \rceil - 1 = \lceil 11.67 \rceil - 1 = 11.)$$

4 PFAIR SCHEDULING OF PINWHEEL TASKS

Given a system of pinwheel tasks Γ, with each task x characterized by computation requirement $x.a$ and window size $x.b$, our goal here is to use Algorithm PF to generate a schedule for Γ. Since Algorithm PF uses the weight of the tasks to make scheduling decisions, it is necessary to first define a weight $x.w$ to each task x, based upon the values of $x.a$ and $x.b$.

Consider the following pinwheel scheduling algorithm:

Algorithm Pinfair(Γ), where Γ is a system of pinwheel tasks.
Step 1: For each $x \in \Gamma$, define a *weight* $x.w$ as follows:

$$x.w \leftarrow \frac{x.a+1}{x.b}.$$

Step 2: If $\sum_{x \in \Gamma} x.w > 1$ return failure.
Step 3: Schedule Γ using Algorithm PF.

THEOREM 2. *Any system of pinwheel tasks* Γ *satisfying*

$$\sum_{x \in \Gamma} x.w \le 1 \tag{3}$$

(*where* $x.w$ *is as defined in Algorithm* Pinfair) *can be successfully scheduled by Algorithm* Pinfair.

PROOF. Observe first that, if the condition of the theorem (3) is satisfied, then Algorithm Pinfair reduces to Algorithm PF with task weights as defined in Step 1. In the remainder of this proof, we therefore study the behavior of Algorithm PF on a task system in which the task weights are as defined in Step 1. Hence, let $x.w = (x.a + 1)/x.b$ for each task x. From the perspective of meeting the pinwheel condition, the "worst" case that can occur during the scheduling of the task system using Algorithm PF is for task x to get scheduled for the jth time at the earliest possible slot and for the next $x.a$ allocations to x to occur as late as possible. In that case, there will be exactly $x.a$ allocations to x over the interval between time-instants (earliest(x, j) + 1) and (latest(x, $j + x.a$) + 1). In order to ensure that Algorithm PF schedules task x for at least $x.a$ of every $x.b$ consecutive slots, it suffices to ensure that the size of this interval never exceeds $x.b$, for any $j \in \mathbf{N}$. That is, for all j,

$$\text{latest}(x, j + x.a) + 1 - (\text{earliest}(x, j) + 1) \le x.b.$$

Notice that

$$\begin{aligned}
&\text{latest}(x, j + x.a) + 1 - \text{earliest}(x, j) - 1 \\
&= \left\lceil \frac{j + x.a + 1}{x.w} \right\rceil - \left\lfloor \frac{j}{x.w} \right\rfloor - 1 \\
&= \left\lceil \frac{j}{x.w} + \frac{x.a+1}{x.w} \right\rceil - \left\lfloor \frac{j}{x.w} \right\rfloor - 1 \\
&= \left\lceil \frac{j}{x.w} \right\rceil + x.b - \left\lceil \frac{j}{x.w} \right\rceil - 1 \quad \text{(By the assumption that } x.w = (x.a+1)/x.b) \\
&\le x.b - 1 + 1 \qquad\qquad \text{(Since } \lceil j/x.w \rceil - \lfloor j/x.w \rfloor \le 1) \\
&= x.b
\end{aligned}$$

Hence, choosing $x.w = (x.a + 1)/x.b$ in Step 1 of Algorithm Pinfair will ensure that Algorithm PF will schedule task x at least $x.a$ times in any interval of size $x.b$. □

EXAMPLE 2. Consider a system of two pinwheel tasks $x_1.a = 5$, $x_1.b = 10$ and $x_2.a = 2$, $x_2.b = 9$, which are to be scheduled on a single processor. To schedule this pinwheel system, Algorithm Pinfair will compute a weight $x_1.w = (5 + 1)/10$ and $x_2.w = (2 + 1)/9$, and call Algorithm PF on the periodic task system with tasks of weights 0.6 and $0.\overline{3}$, respectively. But, this is exactly the system considered in Example 1; hence, the schedule generated there (reproduced below) is also a pinwheel schedule for pinwheel tasks x_1 and x_2:

slot:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
task:	1	2	1	1	2	1	1	2	1	2	1	1	2	1	⊥
slot:	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
task:	1	2	1	1	2	1	1	2	1	2	1	1	2	1	⊥

The reader may verify that every interval of 10 contiguous slots in this schedule contains at least five 1s, and that every interval of nine contiguous slots contains at least two 2s.

5 Evaluation

The "best" currently known algorithm for pinwheel scheduling is the S_{xy} scheduler of Chan and Chin [8]. In this section, we compare the performance of Algorithm Pinfair as a scheduling algorithm for generalized pinwheel task systems with the performance of S_{xy}.

5.1 Density Threshold

The *density threshold* of a scheduling algorithm is defined to be the largest number w such that any pinwheel task system with density at most w can be successfully scheduled by this algorithm. The seminal paper on pinwheel scheduling [15] presented a pinwheel scheduling algorithm with a density threshold of one-half. Chan and Chin [9], [8] subsequently improved this bound, designing a series of algorithms with successively better density bounds, culminating finally in one—S_{xy}—with a density threshold of 0.7 [8]. This threshold is achieved on a task system Γ with $b_{\min} = 3$, where $b_{\min} \stackrel{\text{def}}{=} \min_{x \in \Gamma}\{x.b\}$. The density threshold increases with increasing $b_{\min}$; Chan and Chin conjecture [8, p. 764] that the density threshold approaches (in fact, exceeds) 0.82 as $b_{\min} \to \infty$.

When density threshold is the metric, Algorithm Pinfair compares rather poorly with Scheduler S_{xy}:

LEMMA 1. *Algorithm* Pinfair *has a density threshold of* $1/2$.

PROOF. Algorithm Pinfair assigns a weight of 1 to a task x_1 with $x_1.a = 1$ and $x_1.b = 2$; therefore, it will fail to schedule any pinwheel task system $\{x_1, x_2\}$, where $x_2.a = 1$, $x_2.b = d$ for any $d \in \mathbf{N}$. The density threshold of Algorithm Pinfair is, therefore, no more than one-half.

It follows from Theorem 3 (below) that the density threshold is at least one-half. Hence, the lemma. □

This poor performance of Algorithm Pinfair for unrestricted generalized pinwheel task systems is not very surprising, since it has been explicitly designed for a specific *kind* of task system—those where all the computation requirements (the $x.a$ parameters) are not too small. When this is true, the density threshold of Algorithm Pinfair improves considerably:

THEOREM 3. *Algorithm* Pinfair *successfully schedules any pinwheel task system* Γ *with density at most* $a_{\min}/(a_{\min} + 1)$, *where*

$$a_{\min} \stackrel{\text{def}}{=} \min_{x \in \Gamma}\{x.a\}.$$

PROOF. Let ρ denote the density of Γ. Then

$$\sum_{x \in \Gamma} \frac{x.a + 1}{x.b} = \sum_{x \in \Gamma} \frac{x.a\left(1 + 1/x.a\right)}{x.b}$$
$$\leq \sum_{x \in \Gamma} \frac{x.a\left(1 + 1/a_{\min}\right)}{x.b} = \left(1 + 1/a_{\min}\right)\rho.$$

From Theorem 2, it follows that, in order for Algorithm Pinfair to successfully schedule Γ, it is sufficient that

$$\left(\left(1 + 1/a_{\min}\right)\rho \leq 1\right) \equiv \left(\rho \leq 1/\left(1 + 1/a_{\min}\right)\right)$$
$$\equiv \left(\rho \leq \left(a_{\min}/\left(a_{\min} + 1\right)\right)\right).$$

□

Thus, the density threshold rapidly increases with a_{min} (for $a_{min} = 4$, the density threshold is at least 0.8; for $a_{min} = 5$, it is ≥ 0.825; for $a_{min} = 9$, it is ≥ 0.9), asymptotically approaching unity with increasing a_{min}. (Recall that Chan and Chin conjecture [14, p. 764] that the density threshold of Scheduler S_{xy} is close to 0.82 as $b_{min} \to \infty$; hence, for $a_{min} \geq 5$, the density threshold of Algorithm Pinfair exceeds the (conjectured) threshold of Scheduler S_{xy}.)

5.2 Run-Time Complexity

Steps 1 and 2 of Algorithm Pinfair together take time linear in the size of the input task system Γ.

For task systems described in the original model of Holte et al. [15], Chan and Chin's Scheduler S_{xy} has a preprocessing time of $O(n^2)$, where $n \stackrel{\text{def}}{=} |\Gamma|$. However, it is not clear that S_{xy} can be adapted to *generalized* pinwheel task systems such that the preprocessing time is polynomial in the number of tasks in Γ.

As has been discussed in Section 3, Algorithm PF can be implemented to run in $O(\log n)$ time per slot. Step 3 of Algorithm Pinfair, therefore, takes $O(\log n)$ time per slot. The approach of Chan and Chin [8] specifically aimed to implement their scheduler in constant time per slot (their scheduler was, thus, a "Fast On-Line Scheduler" or a FOLS [15]), but required $O(n)$ processors in order to do so.

5.3 Extension to Multiprocessors

The definition of the generalized pinwheel scheduling problem may be extended to multiple processors in an obvious manner: *Given a multiset* $\{(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$ *of ordered pairs of positive integers and a positive integer* m, *determine an infinite sequence of* m-*subsets over the symbols* $\{1, 2, 3, \ldots, n\}$ *such that, for each* i, $1 \leq i \leq n$, *any subsequence of* b_i *consecutive* m-*subsets has at least* a_i m-*subsets that contain the symbol* i. The interpretation is that there are m identical copies of the resource available and that no task may use more than one copy of the resource at any time.

Algorithm PF was initially defined in the context of multiprocessor scheduling of periodic task systems [5]. Using Algorithm PF, and techniques essentially identical to the ones used in this paper, a multiprocessor variant of Algorithm Pinfair can be defined, which schedules any pinwheel task system Γ on m processors, provided

$$\sum_{x \in \Gamma} x.w \leq m,$$

where $x.w$ is as defined by Algorithm Pinfair. Equivalently, given a generalized pinwheel task system Γ, $\left\lceil \sum_{x \in \Gamma} x.w \right\rceil$ processors are sufficient for scheduling the system by using (the multiprocessor variant of) Algorithm Pinfair.

6 Conclusions

When initially introduced, pinwheel tasks were each characterized by a single parameter—a *window size*. They have since been generalized to enhance their expressive power as a modeling tool, by incorporating a second parameter—a *computation requirement*. While the expressive power of the model has been significantly increased as a result of this generalization, not too much research has gone into the design of scheduling algorithms explicitly built for this new model. In this paper, we have presented and proved correct Algorithm Pinfair, a scheduling algorithm that

1) is extremely efficient in terms of runtime complexity—taking $O(n)$ preprocessing time and $O(\log n)$ time per slot in scheduling a system of n pinwheel tasks,
2) has a superior density threshold to previously known algorithms for the subclass of generalized pinwheel task systems when all the computation requirements are relatively large, and
3) generalizes in a straightforward manner to multiple-processor platforms.

ACKNOWLEDGMENTS

This work has been partially supported by the U.S. National Science Foundation (grants CCR-9796028 and CCR-9704206) and the National Science Council of China (grant NSC-87-2213-E-003-001).

REFERENCES

[1] M. Ajtai, J. Aspnes, M. Naor, Y. Rabani, L. Schulman, and O. Waarts, "Fairness in Scheduling," *Proc. Sixth Ann. ACM-SIAM Symp. Discrete Algorithms*, pp. 477-485, Jan. 1995.

[2] A. Bar-Noy, A. Mayer, B. Schieber, and M. Sudan, "Guaranteeing Fair Service to Persistent Dependent Tasks," *SIAM J. Computing*, vol. 28, no. 4, pp. 1,168-1,189, Aug. 1998.

[3] S. Baruah, "Fairness in Periodic Real-Time Scheduling," *Proc. 16th Real-Time Systems Symp.*, pp. 200-209, Pisa, Italy, 1995.

[4] S. Baruah and A. Bestavros, "Timely and Fault-Tolerant Data Access from Broadcast Disks: A Pinwheel-Based Approach," *Proc. Workshop Databases: Active & Real-Time*, pp. 45-49, Rockville, Md., Nov. 1996.

[5] S. Baruah, N. Cohen, G. Plaxton, and D. Varvel, "Proportionate Progress: A Notion of Fairness in Resource Allocation," *Algorithmica*, vol. 15, no. 6, pp. 600-625, June 1996. Extended Abstract in *Proc. 1993 ACM Ann. Symp. Theory of Computing*.

[6] S. Baruah, L. Rosier, and D. Varvel, "Static and Dynamic Scheduling of Sporadic Tasks for Single-Processor Systems," *Proc. Third Euromicro Workshop Real-time Systems*, June 1991.

[7] S. Baruah, J. Gehrke, and G. Plaxton, "Fast Scheduling of Periodic Tasks on Multiple Resources," *Proc. Ninth Int'l Parallel Processing Symp.*, pp. 280-288. IEEE CS Press, Apr. 1995. Extended version available via anonymous ftp from ftp.cs.utexas.edu as Technical Report TR–95–02.

[8] M.Y. Chan and F. Chin, "Schedulers for the Pinwheel Problem Based on Double-Integer Reduction," *IEEE Trans. Computers*, vol. 41, no. 6, pp. 755-768, June 1992.

[9] M.Y. Chan and F. Chin, "Schedulers for Larger Classes of Pinwheel Instances," *Algorithmica*, vol. 9, no. 5, pp. 425-462, 1993.

[10] M. Dertouzos, "Control Robotics: The Procedural Control of Physical Processors," *Proc. IFIP Congress*, pp. 807-813, 1974.

[11] L. Ford and D. Fulkerson, *Flows in Networks*. Princeton, N.J.: Princeton Univ. Press, 1962.

[12] C.C. Han and K.J. Lin, "Scheduling Distance-Constrained Real-Time Tasks," *Proc. Real-Time Systems Symp.*, pp. 300-308, Dec. 1992.

[13] C.C. Han and K.G. Shin, "A Polynomial Time Optimal Synchronous Bandwidth Allocation Scheme for the Time-Token MAC Protocol," *Proc. IEEE INFOCOM '95*, pp. 875-882, Apr. 1995.

[14] C.C. Han and K.G. Shin, "Real-Time Communication in FieldBus Multiaccess Networks," *Proc. Real-Time Technology and Applications Symp.*, pp. 86-95, May 1995.

[15] R. Holte, A. Mok, L. Rosier, I. Tulchinsky, and D. Varvel, "The Pinwheel: A Real-Time Scheduling Problem," *Proc. 22nd Hawaii Int'l Conf. System Science*, pp. 693-702, Kailua-Kona, Hawaii, Jan. 1989.

[16] R. Holte, L. Rosier, I. Tulchinsky, and D. Varvel, "Pinwheel Scheduling with Two Distinct Numbers," *Theoretical Computer Science*, vol. 100, no. 1, pp. 105-135, 1992.

[17] C.W. Hsueh, K.J. Lin, and N. Fan, "Distributed Pinwheel Scheduling with End-to-End Timing Constraints," *Proc. Real-Time Systems Symp.*, pp. 172-181, Dec. 1995.

[18] S.S. Lin and K.J. Lin, "A Pinwheel Scheduler for Three Distinct Numbers with a Tight Schedulability Bound," *Algorithmica*, vol. 19, no. 4, pp. 411-426, 1997.

[19] C. Liu, "Scheduling Algorithms for Multiprocessors in a Hard Real-Time Environment," *JPL Space Programs Summary 37-60*, vol. II, pp. 28-37, 1969.

[20] C. Liu and J. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment," *J. ACM*, vol. 20, no. 1, pp. 46-61, 1973.

[21] A. Mok, "Task Management Techniques for Enforcing ED Scheduling on a Periodic Task Set," *Proc. Fifth IEEE Workshop Real-Time Software and Operating Systems*, pp. 42-46, Washington, D.C., May 1988.